

# PixelSNE: Visualizing Fast with Just Enough Precision via Pixel-Aligned Stochastic Neighbor Embedding\*

Minjeong Kim  
Korea University  
minjeong1642@gmail.com

Minsuk Choi  
Korea University  
mchoi@korea.ac.kr

Sunwoong Lee  
Korea University  
solomoj94@gmail.com

Jian Tang  
University of Michigan  
jiant@umich.edu

Haesun Park  
Georgia Tech  
hpark@cc.gatech.edu

Jaegul Choo<sup>†</sup>  
Korea University  
jchoo@korea.ac.kr

## ABSTRACT

Embedding and visualizing large-scale high-dimensional data in a two-dimensional space is an important problem since such visualization can reveal deep insights out of complex data. Most of the existing embedding approaches, however, run on an excessively high precision, ignoring the fact that at the end, embedding outputs are converted into coarse-grained discrete pixel coordinates in a screen space. Motivated by such an observation and directly considering pixel coordinates in an embedding optimization process, we accelerate Barnes-Hut tree-based t-distributed stochastic neighbor embedding (BH-SNE), known as a state-of-the-art 2D embedding method, and propose a novel method called PixelSNE, a highly-efficient, screen resolution-driven 2D embedding method with a linear computational complexity in terms of the number of data items. Our experimental results show the significantly fast running time of PixelSNE by a large margin against BH-SNE, while maintaining the minimal degradation in the embedding quality. Finally, the source code of our method is publicly available at <https://github.com/awesome-davian/sasne>.

## Keywords

t-SNE; Barnes-Hut SNE; Dimension reduction; 2D embedding; Scatterplot; Visualization

## 1. INTRODUCTION

Visualizing high-dimensional data as a two-dimensional

\*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

<sup>†</sup>the corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).

WOODSTOCK '97 El Paso, Texas USA

© 2016 ACM. ISBN 978-1-4503-2138-9...\$15.00

DOI: 10.1145/1235

(2D) or three-dimensional (3D)<sup>1</sup> scatterplot is an effective approach that can reveal deep insights about data. Through such visualization, one can obtain the idea about the relationships among clusters as well as those of individual data, e.g., similar or outlying clusters and/or data items.

To generate a scatterplot given high-dimensional data, one can apply various dimension reduction or low-dimensional embedding approaches including traditional methods (e.g., principal component analysis [10] and multidimensional scaling [12, 13]) and recent manifold learning methods (e.g., isometric feature mapping [22], locally linear embedding [20], and Laplacian Eigenmaps [2]).

These methods, however, do not properly handle the significant information loss due to reducing high dimensionality down to two or three, and in response, an advanced dimension reduction technique called t-distributed stochastic neighbor embedding (t-SNE) [23] has been proposed, showing its outstanding advantages in generating 2D scatterplots.

A drawback of t-SNE is its significant computing time against a large number of data items, e.g., the computational complexity of  $O(n^2)$ , where  $n$  is the number of data items. Although various approximation techniques attempting to accelerate its algorithm have been proposed, e.g., Barnes-Hut SNE (BH-SNE) [25] with the complexity of  $O(n \log n)$ , it still takes much time to apply them to large-scale data.

To tackle this issue, this paper proposes the novel framework that can significantly accelerate the 2D embedding algorithms in visualization applications. The proposed framework is motivated by the fact that most embedding approaches compute the low-dimensional coordinates with an excessive precision, e.g., a double precision with 64-bit floating point representation, compared to the pixel coordinates that they are mapped to in our screen space. For instance, if our screen space has  $1024 \times 768$  pixels, then the resulting coordinates computed from an embedding algorithm will be discretized into 1024 and 768 integer levels of  $x$  and  $y$  coordinates, respectively. Moreover, when making sense of a 2D scatterplot, human perception does not often require the results with a high precision [6].

Applying this idea of the just enough precision for our screen and human perception to the above-mentioned state-

<sup>1</sup>This paper discusses only the 2D embedding case, but our proposed approach can be easily extended to the 3D case.

of-the-art method, BH-SNE, we propose a significantly fast 2D embedding approach called pixel-aligned SNE (PixelSNE), which shows more than 2x fold speedup against BH-SNE for 100,000 data item of Yelp datasets. In detail, by lowering and aligning the precision used in BH-SNE to that of pixels in a screen space, PixelSNE directly optimizes those pixel coordinates in the screen that has a pre-given resolution.

In this paper, we describe the mathematical and algorithmic details of how we applied the idea of a pixel-based precision to BH-SNE and present the extensive experimental results that show the significantly fast running time of PixelSNE by a large margin against BH-SNE, while maintaining the minimal degradation in the embedding quality.

Finally, our contributions can be summarized as follows:

1. We present a novel framework of a pixel-based precision driven by a screen space with a finite resolution.
2. We propose a highly-efficient 2D embedding approach called PixelSNE by leveraging our pixel-based precision idea in BH-SNE.
3. We perform extensive quantitative and qualitative analyses using various real-world datasets, showing the significant speedup of our proposed approach against BH-SNE along with a comparable quality of visualization results.

## 2. RELATED WORK

Dimension reduction or low-dimensional embedding of high-dimensional data [27] has long been an active research area. Typically, most of these methods attempt to generate the low-dimensional coordinates that maximally preserve the pairwise distances/similarities given in a high-dimensional space. Such low-dimensional outputs generally work for two purposes: (1) generating the new representations of original data for improving the desired performance of a downstream target task, such as its accuracy and/or computational efficiency, and (2) visualizing high-dimensional data in a 2D scatterplot for providing humans with the in-depth understanding and interpretation about data.

Widely-used dimension reduction methods used for visualization application include principal component analysis (PCA) [10], multidimensional scaling [12, 13], Sammon mapping [19], generative topographic mapping [4], and self-organizing map [11]. While these traditional methods generally focus on preserving global relationships rather than local ones, a class of nonlinear, local dimension reduction techniques called manifold learning [15] has been actively studied, trying to recover an intrinsic curvilinear manifold out of given high-dimensional data. Representative methods are isometric feature mapping [22], locally linear embedding [20], Laplacian Eigenmaps [2], maximum variance unfolding [29], and autoencoder [8].

Specifically focusing on visualization applications, a recent method, t-distributed stochastic neighbor embedding [23], which is built upon stochastic neighbor embedding [9], has shown its superiority in generating the 2D scatterplots that can reveal meaningful insights about data such as clusters and outliers. Since then, numerous approaches have been proposed to improve the visualization quality and its related performances in the 2D embedding results. For example, a neural network has been integrated with t-SNE to learn the parametric representation of 2D embedding [16]. Rather than the Euclidean distance or its derived similarity information, other information types such as non-metric similar-

ities [26] and relative ordering information about pairwise distances in the form of similarity triplets [24] have been considered as the target information to preserve. Additionally, various other optimization criteria and their optimization approaches, such as elastic embedding [5] and NeRV [28], have been proposed.

The computational efficiency and scalability of 2D embedding approaches has also been widely studied. An accelerated t-SNE based on the approximation using the Barnes-Hut tree algorithm has been proposed [25]. Gisbrecht et al. proposed a linear approximation of t-SNE [7]. More recently, an approximate, but user-steerable t-SNE, which provides interactions with which a user can control the degree of approximation on user-specified areas, has also been studied [18]. In addition, a scalable 2D embedding technique called LargeVis [21] significantly reduced the computing times with a linear time complexity in terms of the number of data items.

Even with such a plethora of 2D embedding approaches, to the best of our knowledge, none of the previous studies have directly exploited the limited precision of our screen space and human perception for developing highly efficient 2D embedding algorithms, and our novel framework of controlling the precision in return for algorithm efficiency and the proposed PixelSNE, which significantly improves the efficiency of BH-SNE, can be one such example.

## 3. PRELIMINARIES

In this section, we introduce the problem formulation and the two existing methods, t-distributed stochastic neighbor embedding (t-SNE) and its efficient version called Barnes-Hut SNE (BH-SNE).

### 3.1 Problem Formulation

A 2D embedding method takes a set of high-dimensional data items,  $\mathcal{X} = \{x_i \in \mathbb{R}^D\}_{i=1,2,\dots,N}$ , where  $N$  is the number of data items and  $D$  is the original dimensionality, and gives their 2D (low-dimensional) embedding,  $\mathcal{Y} = \{y_i \in \mathbb{R}^2\}_{i=1,2,\dots,N}$ , as an output. Given a screen of resolution  $r_1 \times r_2$ , where  $r_1$  and  $r_2$ , the  $x$ - and  $y$ -axis resolutions, respectively, are positive integers, the scatterplot is generated by transforming  $\mathcal{Y}$  to their corresponding (zero-based) pixel coordinates

$$\mathcal{Z} = \left\{ \lfloor z_i \rfloor : z_i = \begin{bmatrix} z_{i,1} \\ z_{i,2} \end{bmatrix} \in \mathbb{R}^2, \right\}_{i=1,2,\dots,N} \quad \text{where} \quad (1)$$

$$0 \leq z_{i,d} < r_d, \forall d \in \{1, 2\}. \quad (2)$$

### 3.2 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE embeds high-dimensional data into a low-dimensional space by minimizing the differences between the joint probability distribution representing pairwise relationships in  $\mathcal{X}$  and its counterpart in  $\mathcal{Y}$ . In detail, t-SNE computes the Euclidean pairwise distance matrix  $D_{\mathcal{X}} \in \mathbb{R}^{N \times N}$  of  $\mathcal{X}$  and then converts it to the high-dimensional joint probability matrix  $P \in \mathbb{R}^{N \times N}$  using a Gaussian kernel.

Next, given randomly initialized  $\mathcal{Y}$ , t-SNE computes the Euclidean pairwise distance matrix  $D_{\mathcal{Y}} \in \mathbb{R}^{N \times N}$  and then converts it to a low-dimensional joint probability matrix  $Q \in \mathbb{R}^{N \times N}$  using a Student's  $t$ -distribution. To be specific, the  $(i, j)$ -th component  $q_{ij}^{(t)}$  of  $Q^{(t)}$  at iteration  $t$ , which rep-

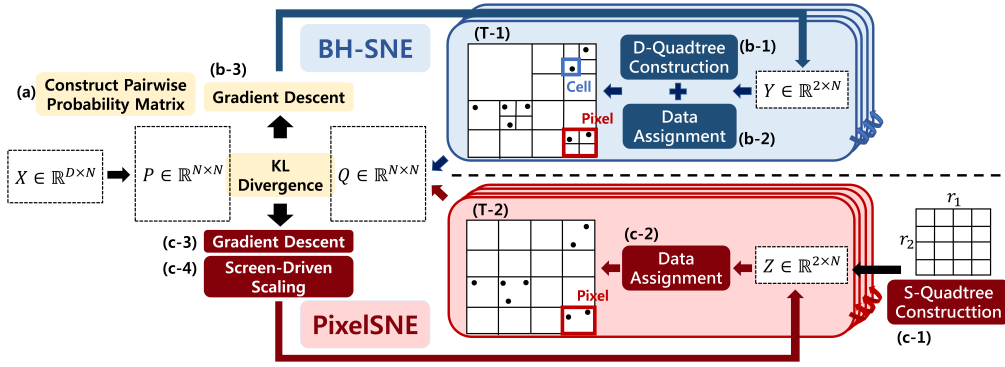


Figure 1: Overview of the proposed PixelSNE in comparison with the original Barnes-Hut SNE (BH-SNE)

resents the similarity between  $y_i^{(t)}$  and  $y_j^{(t)}$  in a probabilistic sense, is computed as

$$q_{ij}^{(t)} = \left( \left( 1 + \|y_i^{(t)} - y_j^{(t)}\|_2^2 \right) Z^{(t)} \right)^{-1} \quad (3)$$

where  $Z^{(t)} = \sum_{k \neq l} \left( 1 + \|y_i^{(t)} - y_j^{(t)}\|_2^2 \right)^{-1}$ .

The objective function of t-SNE is defined using the Kullback-Leibler divergence between  $P$  and  $Q$  as

$$C = KL(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}, \quad (4)$$

and t-SNE iteratively performs gradient-descent update on  $\mathcal{Y}$  where the gradient with respect to  $y_i$  is computed [25] as

$$\begin{aligned} \frac{\partial C}{\partial y_i^{(t)}} = & 4(F_{attr} + F_{rep}) = 4 \left( \sum_{j \neq i} p_{ij} q_{ij}^{(t)} Z^{(t)} \left( y_i^{(t)} - y_j^{(t)} \right) \right. \\ & \left. - \sum_{j \neq i} \left( q_{ij}^{(t)} \right)^2 Z^{(t)} \left( y_i^{(t)} - y_j^{(t)} \right) \right). \end{aligned} \quad (5)$$

Note that every data point exerts an attraction and an repulsion forces to one another based on the difference between the two pairwise joint probability matrices  $P$  and  $Q$ .

### 3.3 Barnes-Hut-SNE (BH-SNE)

While t-SNE adopts a brute-force approach with the computational complexity  $\mathcal{O}(N^2)$  considering all the pairwise relationships, BH-SNE adopts two different tree-based approximation methods to reduce this complexity. The first one called the vantage-point tree [30] approximately computes  $D_{\mathcal{X}}$  and then  $P$  as sparse matrices by ignoring small pairwise distances as zeros (Fig. 1(a)). This approximation reduces the complexity  $\mathcal{O}(N^2)$  of computing  $D_{\mathcal{X}}$  and  $P$  to  $\mathcal{O}(uN \log N)$  where  $u$  is a pre-defined parameter of perplexity and  $u \ll N$ . Accordingly, involving only those nonzero  $p_{ij}$ 's in the sparse matrix  $P$ , the complexity of computing  $F_{attr}$  in Eq. (2) reduces to  $\mathcal{O}(uN)$ .

When it comes to optimizing low-dimensional coordinates (Fig. 1(b)), BH-SNE adopts Barnes-Hut algorithm [1] to compute  $F_{rep}$  in Eq. (5). When  $\|y_i - y_j\| \approx \|y_i - y_k\| \gg \|y_j - y_k\|$ , the forces of  $y_j$  and  $y_k$  to  $y_i$  are similar to each other when computing the gradient. Based on this observation, BH-SNE uses Barnes-Hut algorithm to find a single representative point of multiple data points used for gradient update. For example, if we set the representative data

point of  $y_j$  and  $y_k$  as  $y_s$ , then the gradient  $q_{is}$  can be used to substitute each of  $q_{ij}$  and  $q_{ik}$ .

To dynamically obtain the representative points  $y_s$ 's, BH-SNE constructs a quadtree at each iteration, recursively splitting the 2D region that contains  $y_i$ 's into four pieces. Each node, which we call a cell  $c$ , contains the following information:

1. the boundary  $\mathcal{B}_d^{(c)}$  about its corresponding 2D region, i.e.,

$$\mathcal{B}_d^{(c)} = \left[ b_{min,d}^{(c)}, b_{max,d}^{(c)} \right) \text{ for } d = 1, 2 \quad (6)$$

2. the set  $\mathcal{Y}_c$  of  $y_i$ 's contained in  $c$ , i.e.,

$$\mathcal{Y}_c = \left\{ y_i : y_{i,d} \in \mathcal{B}_d^{(c)} \text{ for } d = 1, 2 \right\}, \quad (7)$$

and its cardinality,  $N_c = |\mathcal{Y}_c|$ , and

3. the center of mass,  $y_c$ , of  $y_i$ 's in  $c$ , i.e.,

$$y_c = \frac{1}{N_c} \sum_{y_i \in \mathcal{Y}_c} y_i \quad (8)$$

Given  $y_i^{(t)}$  at iteration  $t$ , BH-SNE starts by forming a root node  $c_{root}$ , containing all the  $y_i^{(t)}$ 's, by setting  $\mathcal{Y}_{c_{root}} = \mathcal{Y}$  in Eq. (7) and

$$b_{min,d}^{(c_{root})} = \min_i y_i^{(t)} \text{ and} \quad (9)$$

$$b_{max,d}^{(c_{root})} = \max_i y_i^{(t)} + \epsilon, \forall d \in \{1, 2\}, \quad (10)$$

where  $\epsilon$  is a small number in Eq. (6). BH-SNE then recursively splits  $c$  into four (equally-sized) quadrant cells located at "northwest", "northeast", "southwest", and "southeast" by setting their boundaries accordingly, assigning  $y_i^{(t)}$ 's to these child cells based on the boundary information in Eq. (7), and computing their centers of mass. As assigning  $y_i^{(t)}$ 's one by one to the tree, the quadtree grows until each leaf node contains at most a single  $y_i^{(t)}$ .

Afterwards, when computing the gradient with respect to  $y_i$  in Eq. (5) (Fig. 1(b-3)), BH-SNE traverses the quadtree via depth-first search to determine whether  $y_c$  can work as the representative point of those contained in  $c$  based on the criterion

$$r_c / \|y_i - y_c\|_2^2 < \theta, \quad (11)$$

where  $r_c$  represents the diagonal length of the region of  $c$  and  $\theta$  is a threshold parameter. Once finding  $y_c$  satisfying this criterion, the term  $-q_{ij}^2 Z(y_i - y_j)$  in Eq. (5) for those points

contained in  $c$  is approximated as  $-N_c q_{i,c}^2 Z(y_i - y_c)$ , thus reducing the computational complexity of  $F_{rep}$  in Eq. (5) to  $\mathcal{O}(N \log N)$ .

## 4. PIXEL-ALIGNED SNE (PIXELSNE)

In this section, we present our PixelSNE, which significantly reduces the computational time of BH-SNE.

### 4.1 Pixel-Aligned Barnes-Hut Tree

A major novelty of PixelSNE originates from the fact that it directly optimizes the screen-space coordinates  $z_i$  (Eq. (1)) instead of  $y_i$ . The main properties of  $z_i$  is two-fold: (1) the range of  $z_{i,d}$  remains fixed as  $[0, r_d]$  for  $d = 1, 2$  throughout algorithm iterations (Eq. (2)) and (2)  $z_i$  is further discretized as integer coordinates when mapped to a pixel in a screen. Utilizing these characteristics, we accelerate the Barnes-Hut tree construction as the assignment process of  $z_i$ 's to cells, as follows. For clarification, we denote our accelerated quadtree algorithm of PixelSNE as a pixel-aligned quadtree (S-Quadtree) while we call the original quadtree algorithm of BH-SNE as a data-driven quadtree (D-Quadtree).

**One-time tree construction (instead of every iteration).** Unlike BH-SNE, which builds a quadtree from scratch per iteration, the above-mentioned properties of  $z_i$  allow PixelSNE to build S-Quadtree just one time before the main iterations of gradient-descent update and to recycle it during the iterations. That is, PixelSNE pre-computes the full information about (1) the boundary (Eq. (6)) of each cell and (2) its center of mass (Eq. (8)) as follows.

For the boundary information, we utilize the fact that since  $\min_i z_{i,d}^{(t)} = 0$  and  $\max_i z_{i,d}^{(t)} = r_d$  for every iteration, Eqs. (9) and (10) boil down to

$$b_{min,d}^{(c_{root})} = 0 \text{ and} \quad (12)$$

$$b_{max,d}^{(c_{root})} = r_d + \epsilon, \forall d \in \{1, 2\}, \quad (13)$$

which is no longer dependent on iteration  $t$ . This constant boundary of the root node makes those of all the child cells in S-Quadtree constant as well, and based on this idea, PixelSNE pre-computes the boundaries of all the cells in S-Quadtree.

Next, since the boundary of each cell is already determined, we simply approximate the center of mass  $y_c$  as the center location of the cell corresponding region, e.g.,

$$y_{c,d} = \frac{b_{min,d}^{(c)} + b_{max,d}^{(c)}}{2}, \forall d \in \{1, 2\},$$

which is also not dependent on iteration  $t$ .

Once the pre-computation of the above information (Fig. 1(c-1)) finishes, the iterative gradient optimization in PixelSNE simply assigns  $z_i^{(t)}$ 's to these pre-computed cells in S-Quadtree and updates  $\mathcal{Y}_c$  and  $N_c$  (Fig. 1(c-2)). Note that in contrast, BH-SNE iteratively computes all these steps every iteration (Figs. 1(b-1) and (b-2)), which acts as the critical inefficiency compared to PixelSNE.

**Bounded tree depth based on screen resolution.** Considering a typical screen resolution, BH-SNE performs an excessively precise computation. That is, when mapping D-Quadtree to pixel grids of our screen, one can view that BH-SNE subdivides the pixels, the minimum unit of the screen, into much smaller cells until each leaf cell contains at most one data point (Fig. 1(T-1)).

On the contrary, the minimum size of the cell in S-Quadtree is bounded to the pixel size to avoid an excessive precision in computation (Fig. 1(T-2)). In detail, S-Quadtree keeps splitting the cell  $c$  while satisfying the condition,

$$b_{max,d}^{(c)} - b_{min,d}^{(c)} > 1, \exists d \in \{1, 2\}, \quad (14)$$

which indicates that the cell length is larger than the unit pixel size 1 for at least one of the two axes. As a result, the depth of S-Quadtree is bounded by

$$\max_{d \in \{1, 2\}} \lceil \log_2 r_d \rceil. \quad (15)$$

Such a bounded depth of S-Quadtree causes a leaf node to possibly contain multiple data points, and in the gradient-descent update step, we may not find the cell satisfying Eq. (11) even after reaching a leaf node in the depth-first search traversal. In this case, we just stop the traversal and use the center of mass of the corresponding leaf node to approximately compute  $F_{rep}$  (Eq. (5)) of those points contained in the node.

**Computational complexity.** The depth of the Barnes-Hut tree acts as a critical factor in algorithm efficiency since both the assignment of data point to cells and the approximation of  $F_{rep}$  (Eq. (5)) are performed based depth-first search traversal, the computational complexity of which is linear to the tree depth. In the worst case where a majority of data points are located in a relatively small region in a 2D space and a few outliers are located far from them, D-Quadtree can grow as deeply as the depth of  $N$ , which is much larger than that of S-Quadtree,  $\max_{d \in \{1, 2\}} \lceil \log_2 r_d \rceil$ , when  $r_d \ll N, \forall d \in \{1, 2\}$ . Owing to this characteristics, the overall computational complexity of PixelSNE is obtained as  $\mathcal{O}(N \times \max_{d \in \{1, 2\}} \lceil \log_2 r_d \rceil)$ , which reduces to the linear computational complexity in terms of the number of data items, given the fixed screen resolution  $r_d$ 's.

Intuitively, BH-SNE traverses much finer-grained nodes to obtain the excessively precise differentiation among data points. On the other hand, PixelSNE assumes that as long as multiple data points are captured within a single pixel, they are close enough to be represented as the center of mass for their visualization in a screen. This guarantees the depth of S-Quadtree to be limited by the screen resolution, instead of being influenced by the undesirable non-uniformity of the 2D embedding results during algorithm iterations.

### 4.2 Screen-Driven Scaling

In order for the outputs  $z_i$ 's at every iteration to satisfy Eq. (2) after their gradient-descent update, we scale them as follows. Let us denote  $z_i^{(t-1)}$  as the 2D coordinates computed at iteration  $t-1$  and  $\hat{z}_i^{(t)}$  as its updated coordinates at the next iteration  $t$  via a gradient-descent method. Note that  $z_i^{(t-1)}$  satisfies Eq. (2) while  $\hat{z}_i^{(t)}$  does not. Hence, we normalize each of the 2D coordinates of  $\hat{z}_i^{(t)}$  and obtain

$$z_i^{(t)} = \begin{bmatrix} z_{i,1} \\ z_{i,2} \end{bmatrix} \text{ as}$$

$$z_{i,d}^{(t)} = \frac{r_d \left( \hat{z}_{i,d}^{(t)} - \min_i \hat{z}_{i,d}^{(t)} \right)}{\gamma_d^{(t)}}, \forall d \in \{1, 2\},$$

where  $\gamma_d^{(t)} = \max_i \hat{z}_{i,d}^{(t)} - \min_i \hat{z}_{i,d}^{(t)} + \epsilon$  and  $\epsilon$  is a small constant, e.g.,  $10^{-6}$ . The reason for introducing  $\epsilon$  is to have the

integer-valued 2D pixel coordinates  $[z_i]$  in Eq. (1) lie exactly between 0 and  $(r_d - 1)$  for  $d = 1, 2$ . For example, for a  $1024 \times 768$  resolution, we impose the pixel coordinates at each iteration to exactly have the range from 0 to 1023 and that from 0 to 767 for  $x$ - and  $y$ -coordinates, respectively.

This scaling step, however, affects the computation of the low-dimensional pairwise probability matrix  $Q$  since it scales each pairwise Euclidean distance,  $\|y_i - y_j\|$ , in Eq. (3), resulting in a different probability distribution from the case with no scaling. To compensate this scaling effect in computing  $Q$ , we re-define  $Q$  with respect to the scaled  $z_i$ 's as

$$q_{ij}^{(t)} = \left( \left( 1 + \sum_{d=1}^2 \beta_d^{(t)} \left( z_{i,d}^{(t-1)} - z_{j,d}^{(t-1)} \right)^2 \right) Z^{(t)} \right)^{-1} \quad (16)$$

where  $Z^{(t)} = \sum_{k \neq l} \left( 1 + \sum_{d=1}^2 \beta_d^{(t)} \left( z_{i,d}^{(t-1)} - z_{j,d}^{(t-1)} \right)^2 \right)^{-1}$  and  $\beta_d$  is defined as

$$\beta_d^{(t)} = \prod_{s=1}^{t-1} \left( \frac{\gamma_d^{(s)}}{r_d} \right)^2, \quad \forall d \in \{1, 2\}. \quad (17)$$

By introducing  $\beta_d^{(t)}$  defined in this manner, PixelSNE uses Eq. (16), which is still equivalent to Eq. (3).

### 4.3 Accelerating Construction of $P$

To accelerate the process of constructing the matrix  $P$ , we adopt a recently proposed, highly efficient algorithm of constructing the approximate  $k$ -nearest neighbor graph (K-NNG) [21]. This algorithm builds on top of the classical state-of-the-art algorithm based on random-projection trees but significantly improves its efficiency. It first starts with a few random projection trees to construct an approximate K-NNG. Afterwards, based on the intuition that ‘‘the neighbors of my neighbors are also likely to be my neighbors,’’ the algorithm iteratively improves the accuracy of K-NNG by exploring the neighbors of neighbors defined according to the current K-NNG. In our experiments, we show that this algorithm is indeed able to significantly accelerate the process of constructing  $P$ .

### 4.4 Minor Improvements in Implementation

The implementation of PixelSNE is built upon that of BH-SNE publicly available at <https://github.com/lvdmaaten/bhtsne>. Other than the above algorithmic improvements, we made minor improvements as follows.

First, the Barnes-Hut tree involves many computations of division by two (or powers of two). We replaced such computations by more efficient, bitwise left-shift operations. Similarly, we replaced modulo-two operations, which is used in data point assignment to cells, with bitwise masking with  $0 \times 0001$ . Finally, we replaced the ‘pow’ function from ‘math’ library in C/C++, which is used for squared-distance computation with a single multiplication, e.g.,  $x \times x$  instead of ‘pow( $x, 2$ )’. Although not significant, these minor modifications rendered consistently faster computing times than the case without them.

## 5. EXPERIMENTS

In this section, we present the quantitative analyses as well as qualitative visualization examples of PixelSNE in comparison with original t-SNE and BH-SNE.

All the experiments were conducted on a single desktop computer with dual Intel Xeon E5-2650 processors, 48GB RAM, and Ubuntu 16.04.1 LTS installed.

## 5.1 Experimental Setup

This section describes our experimental setup including datasets and compared methods.

### 5.1.1 Datasets

For our experiments, we selected four real-world datasets as follows: (1) **MNIST** digit images, (2) Facial expression images (**FaceExp**), (3) 20 Newsgroups documents (**20News**), and (4) **Yelp** reviews.

**MNIST**<sup>2</sup> dataset contains a set of 70,000 gray-scale handwritten digit images with  $28 \times 28 = 784$  pixels, resulting in an input matrix  $\mathcal{X}$  of the size  $784 \times 70,000$ . Each image has its digit label among ten different digits.

**FaceExp**<sup>3</sup> dataset contains a set of 28,709 gray-scale facial image data with  $48 \times 48$  pixels, with the label information out of seven different facial expression categories. For this dataset, we extracted the features from each image using a convolutional neural network. This network starts with four convolutional layers with the 32, 64, 96, and 128 filters, respectively, with the size of each filter being  $3 \times 3$ . Each convolutional layer is followed by the batch normalization layer, rectified linear unit (ReLU), and the max-pooling layer over  $2 \times 2$  patches. After then, two fully connected layers follow, the first one with 256 output nodes followed by ReLU and the second with 256 output nodes followed by a softmax classifier for seven facial expression categories. After training this model, we used the last hidden layer output of 256 dimensions as the feature vector of each image, resulting in an input matrix  $\mathcal{X}$  of the size  $256 \times 28,709$ .

**20News**<sup>4</sup> dataset is a collection of 18,846 newsgroup posts on 20 different newsgroup categories. We used the label of each document as one of seven higher-level newsgroup categories. As the input vectors, we used a bag-of-words representation of each document, resulting in an input matrix  $\mathcal{X}$  of the size  $134,410 \times 18,846$ .

**Yelp**<sup>5</sup> review dataset contains around 2.7 million reviews written by 687 thousand users. We pre-processed this dataset by removing the keywords appearing in less than 20 documents and some general keywords such as food, restaurant, etc. After forming the term-document matrix, we randomly selected ten thousand reviews from it, resulting in an input matrix  $\mathcal{X}$  of the size  $19,899 \times 100,000$ . In addition, we extracted topic labels out of ten topics computed by nonnegative matrix factorization [14] as a topic modeling method.

For all datasets, we reduced the dimensionality of each input matrix to 50 by applying PCA, which is a standard pre-processing step of t-SNE and BH-SNE.

### 5.1.2 Compared Methods

We compared our methods against the original t-SNE and BH-SNE. For both methods, we used the publicly available code written by the original author.<sup>6</sup> As their parameters, we used the default parameter values for both methods, e.g.,

<sup>2</sup><http://cs.nyu.edu/~roweis/data.html>

<sup>3</sup><https://goo.gl/W3Z8qZ>

<sup>4</sup><http://qwone.com/~jason/20Newsgroups/>

<sup>5</sup>[https://www.yelp.com/dataset\\_challenge](https://www.yelp.com/dataset_challenge)

<sup>6</sup><https://lvdmaaten.github.io/tsne/>

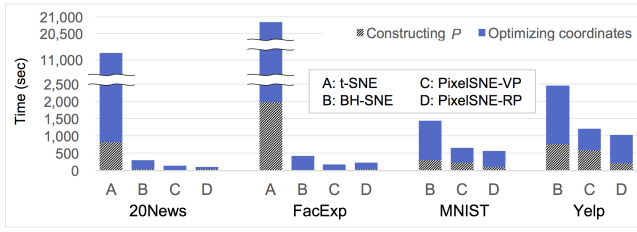


Figure 2: Comparison of computing time. The results of t-SNE are excluded due to its significant computing time.

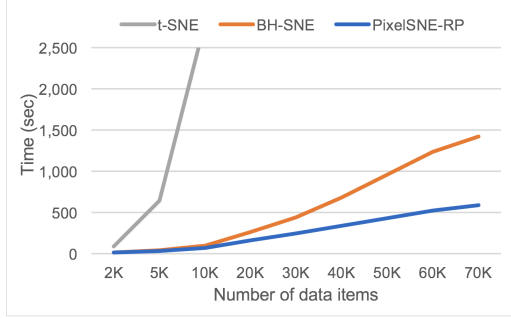


Figure 3: Computing times vs. the number of data items on MNIST dataset

the perplexity value as 50, the number of iterations as 1,000, and the threshold  $\theta$  in Eq. 11 as 0.5.

For PixelSNE, we prepared for its two different versions depending on the algorithm used in constructing  $P$ : (1) the vantage-point tree (**PixelSNE-VP**) used originally in BH-SNE and (2) the random-projection tree (**PixelSNE-RP**) used in LargeVis [21] (Section 4.3). For the latter, we extracted the corresponding part called  $k$ -nearest neighbor construction from the publicly available code<sup>7</sup> and integrated it with our implementation of PixelSNE with its default parameter settings. For the number of iterations and  $\theta$ , we set them as the same as BH-SNE, e.g., 1,000 and 0.5, respectively. For the screen resolution parameters  $r_1$  and  $r_2$ , we set both of them as 512 unless otherwise stated.

## 5.2 Computing Time Results

Fig. 2 shows the comparison of the algorithm efficiency in computing times for different datasets. In all cases, it is shown that PixelSNE-VP and PixelSNE-RP consistently outperform t-SNE and BH-SNE by a large margin. For the part of iteratively optimizing the low-dimensional coordinates, where we mainly applied our screen resolution-driven precision, PixelSNE-RP and PixelSNE-VP both show the significant performance boost against BH-SNE. For instance, for Yelp dataset, this part took 616 and 827 seconds for PixelSNE-VP and PixelSNE-RP, respectively, while it took 1,695 seconds for BH-SNE.

In this case, the computing time difference between PixelSNE-VP and PixelSNE-RP is due to the higher sparsity in the pairwise probability matrix  $P$  of the vantage-point (VP) tree than that of the random-projection (RP) tree, which results

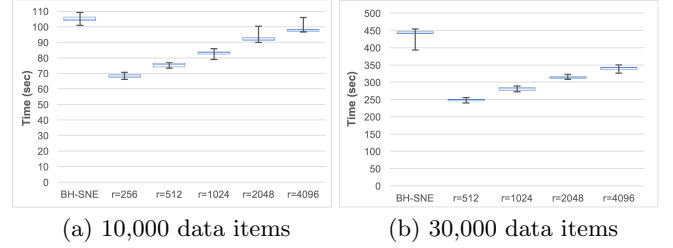


Figure 4: Computing times of PixelSNE-RP vs. the screen resolution  $r$  on MNIST dataset

in a larger amount of computations in  $F_{attr}$  in Eq. (5). We conjecture the reason for denser  $P$  of the RP tree than the VP tree is because the RP tree's approximately finding  $k$  nearest neighbors generated a more asymmetric structure in the pairwise probability matrix, e.g., the case where data  $i$  is a neighbor of data  $j$  but not the other way around, resulting in a denser symmetrized matrix  $P$  than that generated by the VP tree.

However, for the part of constructing  $P$ , as the size of the data gets larger, e.g., MNIST and Yelp datasets, PixelSNE-RP performs significantly better than PixelSNE-VP owing to the highly efficient approach adopted in PixelSNE, as discussed in Section. 4.3.

Next, Fig. 3 shows the computing times depending on the total number of data items, sampled from MNIST dataset. As the data size gets larger, our PixelSNE-RP shows increasingly faster performances than BH-SNE as well as t-SNE, which shows the promising scalability of PixelSNE.

Finally, Fig. 4 shows the computing time depending on the screen resolution parameter  $r$  for 10,000 and 30,000 random samples from MNIST dataset. Although PixelSNE-RP is consistently more efficient than BH-SNE, it tends to run slowly as  $r$  increases. However, from the comparison between Figs. 4(a) and (b), as the data size increases, the computing time of PixelSNE-RP stays low compared to that of BH-SNE for a large values of  $r$ , e.g., 4,096. In practice, we found that the value of  $r$  of more than 4,096 is hardly needed, considering the resolution of the screen size of a commodity monitor.

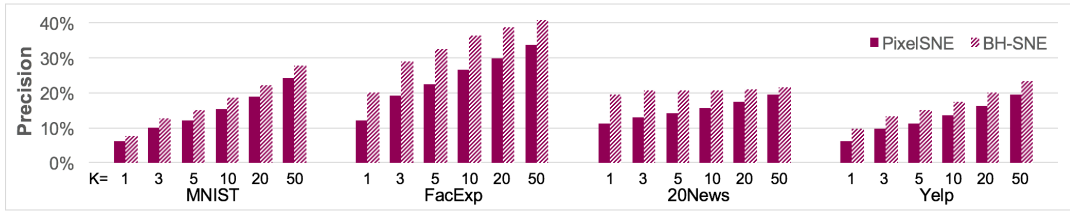
## 5.3 Embedding Quality Results

$r$	$k$ in $k$ nearest neighbors					
	1	3	5	10	20	30
512	.2401 (.0125)	.3207 (.0117)	.3475 (.0098)	.3719 (.0064)	.3916 (.0033)	.4237 (.0023)
1024	.2405 (.0084)	.3165 (.0077)	.3434 (.0061)	.3702 (.0037)	.3932 (.0023)	.4290 (.0011)
2048	.2463 (.0056)	.3216 (.0045)	.3481 (.0046)	.3750 (.0030)	.3977 (.0022)	.4328 (.0008)
4096	.2517 (.0054)	.3272 (.0052)	.3532 (.0046)	.3782 (.0027)	.4006 (.0018)	.4344 (.0009)

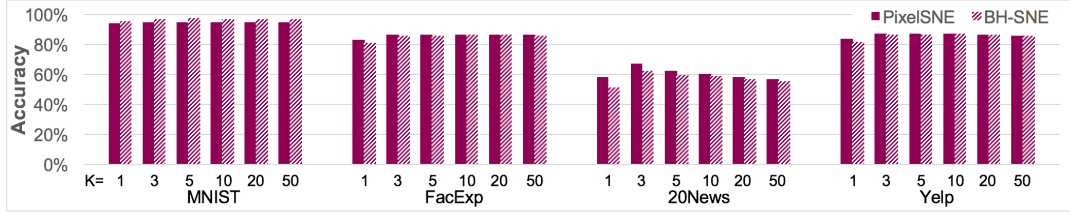
Table 1: Neighborhood precision of PixelSNE-RP on the 10,000 samples of MNIST dataset. The numbers in parentheses represent the standard deviation from ten repetitions.

To quantitatively evaluate the 2D embedding quality, we

<sup>7</sup><https://github.com/lferry007/LargeVis>

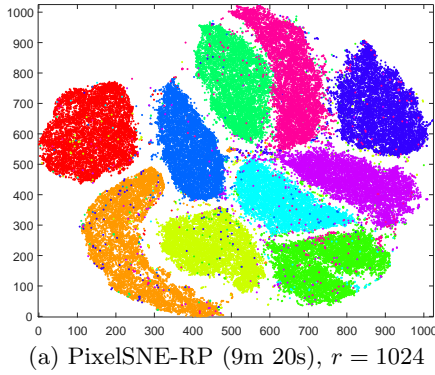


(a) Neighborhood precision

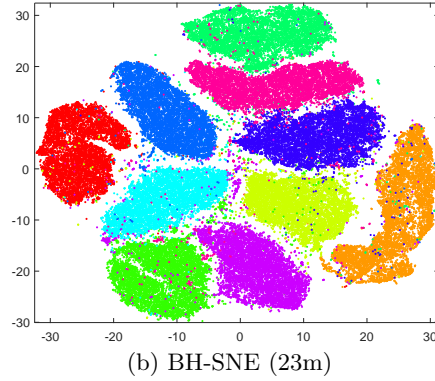


(b)  $k$ -nearest neighbor classification accuracy

Figure 5: Quantitative comparisons of the 2D embedding quality.  $k$  represents the number of the nearest neighbors.



(a) PixelSNE-RP (9m 20s),  $r = 1024$



(b) BH-SNE (23m)

Figure 6: 2D embedding of MNIST dataset. The numbers in parentheses indicate the computing time.

$r$	$k$ in $k$ nearest neighbors					
	1	3	5	10	20	30
512	.9390 (.0012)	.9482 (.0023)	.9463 (.0023)	.9428 (.0023)	.9367 (.0033)	.9297 (.0045)
1024	.9387 (.0017)	.9489 (.0023)	.9475 (.0014)	.9450 (.0014)	.9400 (.0011)	.9335 (.0017)
2048	.9386 (.0020)	.9489 (.0018)	.9472 (.0011)	.9395 (.0020)	.9450 (.0014)	.9323 (.0015)
4096	.9380 (.0018)	.9484 (.0007)	.9471 (.0013)	.9437 (.0013)	.9382 (.0014)	.9324 (.0019)

Table 2:  $k$ -NN classification accuracy of PixelSNE-RP on the 10000 samples of MNIST dataset. The numbers in parentheses represent the standard deviation from ten repetitions.

performed the experiments based on the two following measures:

**Neighborhood precision** measures how the  $k$  original nearest neighbors in the high-dimensional space are preserved or captured in the  $k$  nearest neighbors in the 2D embedding results. In detail, for data  $i$ , let us denote the  $k$  nearest

	BH-SNE	PixelSNE-RP (with $r$ )			
		512	1024	2048	4096
Cost (Eq. 4)	1.815 (.0071)	1.820 (.0303)	1.865 (.0167)	1.871 (.0113)	1.868 (.0167)

Table 3: Comparison of cost function values between BH-SNE and PixelSNE-RP. The numbers in parentheses represent the standard deviation from ten repetitions.

neighbors of data  $i$  in the high-dimensional space and those in the low-dimensional (2D) space as  $\mathcal{N}_D^{(k)}(i)$  and  $\mathcal{N}_d^{(k)}(i)$ , respectively. The neighborhood precision is computed as  $\frac{1}{kN} \sum_{i=1}^N |\mathcal{N}_D^{(k)}(i) \cap \mathcal{N}_d^{(k)}(i)|$ .

**$k$ -NN classification accuracy** measures the  $k$ -nearest neighbor classification accuracy based on the 2D embedding results and their labels.

Fig. 5 shows the results of these two measures depending on different  $k$  values. PixelSNE-RP experienced some degradation in terms of the neighborhood precision compared to BH-SNE (Fig. 5(a)), but it demonstrates comparable  $k$ -NN classification accuracy results to those of BH-SNE for all datasets (Fig. 5(b)). It indicates that given that the  $k$ -NN



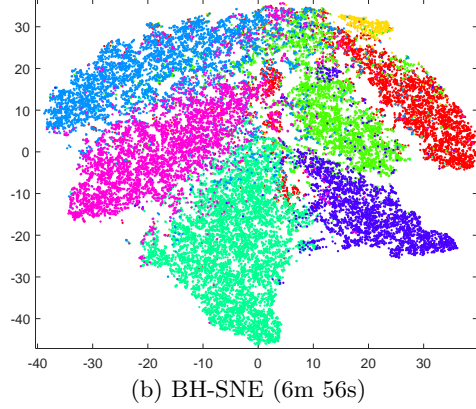
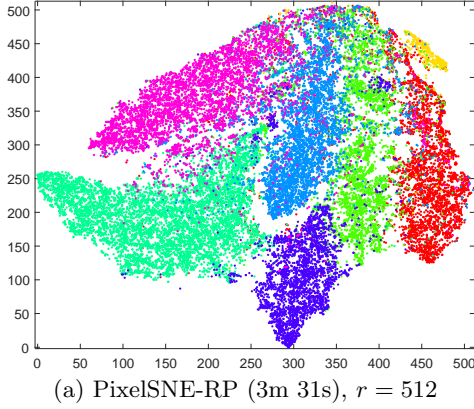


Figure 7: 2D embedding of FacExp dataset. The numbers in parentheses indicate the computing time.

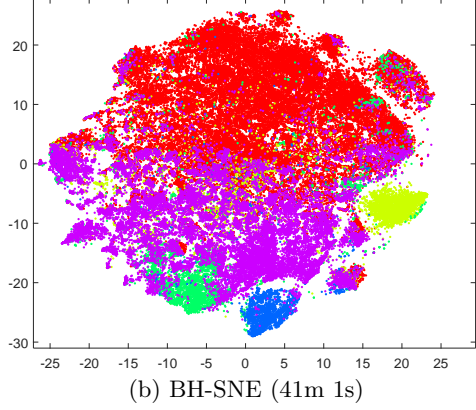
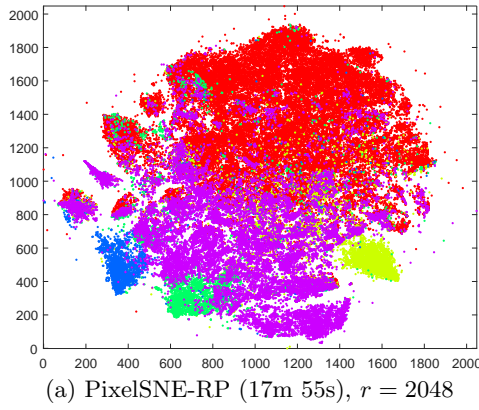


Figure 8: 2D embedding of Yelp dataset. The numbers in parentheses indicate the computing time.

classification accuracy reflects the class separability in the 2D embedding, PixelSNE-RP achieves just enough precision for the screen and human perception, providing no worse quality of the scatterplot visualization by trading off the excessive precision against significant computing time.

On the other hand, Tables 1 and 2 show the above two measures for PixelSNE-RP with respect to different values of a screen resolution  $r$ . Unlike the  $k$ -NN classification accuracy, which stays roughly the same regardless of different values of  $r$ , the neighborhood precision consistently increases as  $r$  gets large. This implies that by controlling  $r$ , one can balance between the embedding quality and the computing time in a user-driven manner.

Finally, we compared the cost function value (Eq. (4)) after convergences, as shown in Table 3. Throughout all the values of  $r$  tested, this value remains almost the same between BH-SNE and PixelSNE-RP, which indicates that the screen resolution-driven precision of PixelSNE has minimal or no impact.

## 5.4 Visualization Examples

Figs. 6-8 shows the 2D embedding examples of several datasets we used.<sup>8</sup> For PixelSNE, we report only the results

<sup>8</sup>Due to the page limit, we excluded the results of 20News dataset.

of PixelSNE-RP since PixelSNE-VP shows much similar results to them. The labels associated with each dataset were used to color-code individual data points. In all these visualization examples, both PixelSNE-RP and BH-SNE demonstrate a comparable quality of results, putting PixelSNE as a promising approach in large-scale visualization.

## 6. CONCLUSIONS

In this paper, we presented a novel idea of exploiting the screen resolution-driven precision as the fundamental framework for significantly accelerating the 2D embedding algorithms, and applying this idea to a state-of-the-art method called Barnes-Hut SNE, we proposed a novel approach called PixelSNE as a highly efficient alternative.

One limitation of our framework is that when zooming in a particular part of a given 2D scatterplot, our method may suffer from the low-precision issue since the results are computed with the precision just enough for a given screen resolution. However, we could overcome this issue via a new algorithm that can dynamically refine the embedding results in real time with a higher precision in a zoomed-in part. As pointed out previously [6], such types of algorithms have connections to well-known existing literature in other fields, such as wavelet transform [17] and adaptive mesh refinement [3], which opens up new research directions



on supporting zoom-in/out capabilities in those embedding algorithms driven by a dynamic, user-driven precision.

Additionally, we plan to apply our framework to other advanced algorithms such as LargeVis [21]. In addition, we plan to develop a parallel distributed version of PixelSNE to further improve the computational efficiency.

## 7. REFERENCES

- [1] Josh Barnes and Piet Hut. A hierarchical  $O(n \log n)$  force-calculation algorithm. *nature*, 324(6096):446–449, 1986.
- [2] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.
- [3] Marsha J Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3):484 – 512, 1984.
- [4] Christopher M Bishop, Markus Svensén, and Christopher KI Williams. GTM: The generative topographic mapping. *Neural computation*, 10(1):215–234, 1998.
- [5] Miguel A Carreira-Perpinán. The elastic embedding algorithm for dimensionality reduction. In *Proc. the International Conference on Machine Learning (ICML)*, pages 167–174, 2010.
- [6] Jaegul Choo and Haesun Park. Customizing computational methods for visual analytics with big data. *IEEE Computer Graphics and Applications (CG&A)*, 33(4):22–28, 2013.
- [7] A. Gisbrecht, B. Mokbel, and B. Hammer. Linear basis-function t-sne for fast nonlinear dimensionality reduction. In *Proc. the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2012.
- [8] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
- [9] Geoffrey E Hinton and Sam T Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 833–840, 2002.
- [10] Ian T. Jolliffe. *Principal component analysis*. Springer, 2002.
- [11] T. Kohonen. *Self-organizing maps*. Springer, 2001.
- [12] J. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29:1–27, 1964.
- [13] J. Kruskal. Nonmetric multidimensional scaling: A numerical method. *Psychometrika*, 29:115–129, 1964.
- [14] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.
- [15] John A Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.
- [16] Laurens Maaten. Learning a parametric embedding by preserving local structure. In *Proc. the International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 5, pages 384–391, 2009.
- [17] S. G. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 11(7):674–693, 1989.
- [18] N. Pezzotti, B. Lelieveldt, L. van der Maaten, T. Holtt, E. Eisemann, and A. Vilanova. Approximated and user steerable tsne for progressive visual analytics. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, PP(99):1–1, 2016.
- [19] Jr. Sammon, John W. A nonlinear mapping for data structure analysis. *Computers, IEEE Transactions on*, C-18(5):401 – 409, may. 1969.
- [20] L. K. Saul and S. T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research (JMLR)*, 4:119–155, 2003.
- [21] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing large-scale and high-dimensional data. In *Proc. the International Conference on World Wide Web (WWW)*, pages 287–297, 2016.
- [22] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.
- [23] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research (JMLR)*, 9:2579–2605, 2008.
- [24] L. van der Maaten and K. Weinberger. Stochastic triplet embedding. In *IEEE International Workshop on Machine Learning for Signal Processing*, pages 1–6, 2012.
- [25] Laurens Van Der Maaten. Accelerating t-sne using tree-based algorithms. *Journal of machine learning research (JMLR)*, 15(1):3221–3245, 2014.
- [26] Laurens van der Maaten and Geoffrey Hinton. Visualizing non-metric similarities in multiple maps. *Machine Learning*, 87(1):33–55, 2012.
- [27] LJP Van der Maaten, EO Postma, and HJ Van den Herik. Dimensionality reduction: A comparative review. *Technical Report TiCC TR 2009-005*, 2009.
- [28] Jarkko Venna, Jaakko Peltonen, Kristian Nybo, Helena Aidos, and Samuel Kaski. Information retrieval perspective to nonlinear dimensionality reduction for data visualization. *Journal of Machine Learning Research (JMLR)*, 11(Feb):451–490, 2010.
- [29] Kilian Weinberger and Lawrence Saul. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70:77–90, 2006.
- [30] Peter N Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, number 194, pages 311–21, 1993.